Study guide: Nonlinear differential equation problems

Hans Petter Langtangen[1,2]

Center for Biomedical Computing, Simula Research Laboratory[1]

Department of Informatics, University of Oslo[2]

Nov 6, 2014

What makes a differential equations nonlinear?

- In linear differential equations, the unknown u or its derivatives appear in linear terms $au(t)$, $au'(t)$, $a\nabla^2 u$, where a is independent of u.
- All other types of terms containing u are *nonlinear* and contain products of u or its derivatives.

examples on linear and nonlinear differential equations

Linear ODE:

$$u'(t) = a(t)u(t) + b(t)$$

Nonlinear ODE:

$$u'(t) = u(t)(1 - u(t)) = u(t) - \color{red}{u(t)^2}$$

This (pendulum) ODE is also nonlinear:

$$u'' + \gamma \sin u = 0$$

because

$$\sin u = u - \frac{1}{6}u^3 + \mathcal{O}(u^5),$$

contains products of u

1. Introduction of basic concepts

2. Systems of nonlinear algebraic equations

3. Linearization at the differential equation level

Introduction of basic concepts

- Logistic ODE as simple model for a nonlinear problem
- Introduction of basic techniques:
  - Explicit time integration (no nonlinearities)
  - Implicit time integration (nonlinearities)
  - Linearization and Picard iteration
  - Linearization via Newton's method
  - Linearization via a trick like geometric mean
- Numerical illustration of the performance

The scaled logistic ODE

$$u'(t) = u(t)(1 - u(t)) = u - u^2$$

Linearization by explicit time discretization

Forward Euler method:

$$\frac{u^{n+1}-u^n}{\Delta t} = u^n(1-u^n)$$

gives a *linear* algebraic equation for the unknown value u^{n+1}!

Explicit time integration methods will (normally) linearize a nonlinear problem.

Another example: 2nd-order Runge-Kutta method

$$\begin{aligned} u^* &= u^n + \Delta t u^n(1-u^n), \\ u^{n+1} &= u^n + \Delta t\frac{1}{2}\left(u^n(1-u^n) + u^*(1-u^*)\right) . \end{aligned}$$

An implicit method: Backward Euler discretization

A backward time difference

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^n)$$

gives a *nonlinear* algebraic equation for the unknown u^n. The equation is of quadratic type (which can easily be solved exactly):

$$\Delta t(u^n)^2 + (1 - \Delta t)u^n - u^{n-1} = 0$$

Detour: new notation

To make formulas less overloaded and the mathematics as close as possible to computer code, a new notation is introduced:

- $u^{(1)}$ means u^{n-1}
- In general: $u^{(\ell)}$ means $u^{n-\ell}$
- u is the unknown (u^n)

Nonlinear equation to solve in new notation:

$$F(u) = \Delta t u^2 + (1 - \Delta t)u - u^{(1)} = 0$$

Exact solution of quadratic nonlinear equations

Solution of $F(u) = 0$:

$$u = \frac{1}{2\Delta t}\left(-1 - \Delta t \pm \sqrt{(1-\Delta t)^2 - 4\Delta t u^{(1)}}\right)$$

Observation:

Nonlinear algebraic equations may have multiple solutions!

How do we pick the right solution in this case?

Let's investigate the nature of the two roots:

```
>>> import sympy as sp
>>> dt, u_1, u = sp.symbols('dt u_1 u')
>>> r1, r2 = sp.solve(dt*u**2 + (1-dt)*u - u_1, u)  # find roots
>>> r1
(dt - sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> r2
(dt + sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> print r1.series(dt, 0, 2)
-1/dt + 1 - u_1 + dt*(u_1**2 - u_1) + O(dt**2)
>>> print r2.series(dt, 0, 2)
u_1 + dt*(-u_1**2 + u_1) + O(dt**2)
```

The `r1` root behaves as $1/\Delta t \rightarrow \infty$ as $\Delta t \rightarrow 0$! Therefore, only the `r2` root is of relevance.

Linearization

- In general, we cannot solve nonlinear algebraic equations with formulas
- We must *linearize* the equation, or create a recursive set of *linearized* equations whose solutions hopefully converge to the solution of the nonlinear equation
- Manual linearization may be an art
- Automatic linearization is possible (cf. Newton's method)

Examples will illustrate the points!

Picard iteration

Nonliner equation from Backward Euler scheme for logistic ODE:

$$F(u) = au^2 + bu + c = 0$$

Let u^- be an available approximation of the unknown u.

Linearization of u^2: $u^- u$

$$F(u) \approx \hat{F}(u) = au^- u + bu + c = 0$$

But

- Problem: the solution u of $\hat{F}(u) = 0$ is not the exact solution of $F(u) = 0$
- Solution: set $u^- = u$ and repeat the procedure

The algorithm of Picard iteration

At a time level, set $u^- = u^{(1)}$ (solution at previous time level) and iterate:

$$u = -\frac{c}{au^- + b}, \quad u^- \leftarrow u$$

This technique is known as

- fixed-point iteration
- successive substitutions
- nonlinear Richardson iteration
- Picard iteration

The algorithm of Picard iteration with classical math notation

- u^k: computed approximation in iteration k
- u^{k+1} is the next approximation (unknown)

$$au^k u^{k+1} + bu^{k+1} + c = 0 \quad \Rightarrow \quad u^{k+1} = -\frac{c}{au^k + b}, \quad k = 0, 1, \ldots$$

Or with a time level n too:

$$au^{n,k} u^{n,k+1} + bu^{n,k+1} - u^{n-1} = 0 \quad \Rightarrow \quad u^{n,k+1} = \frac{u^n}{au^{n,k} + b}, \quad k = 0, 1, \ldots$$

Stopping criteria

Using change in solution:

$$|u - u^-| \leq \epsilon_u$$

or change in residual:

$$|F(u)| = |au^2 + bu + c| < \epsilon_r$$

A single Picard iteration

Common simple and cheap technique: perform 1 single Picard iteration

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^{n-1})$$

Inconsistent time discretization ($u(1-u)$ must be evaluated for n, $n-1$, or $n-\frac{1}{2}$) - can produce quite inaccurate results, but is very popular.

Implicit Crank-Nicolson discretization

Crank-Nicolson discretization:

$$[D_t u = u(1-u)]^{n+\frac{1}{2}}$$

$$\frac{u^{n+1}-u^n}{\Delta t} = u^{n+\frac{1}{2}} - (u^{n+\frac{1}{2}})^2$$

Approximate $u^{n+\frac{1}{2}}$ as usual by an arithmetic mean,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1})$$

$$(u^{n+\frac{1}{2}})^2 \approx \frac{1}{4}(u^n + u^{n+1})^2 \quad \text{(nonlinear term)}$$

which is nonlinear in the unknown u^{n+1}.

Linearization by a geometric mean

Using a *geometric mean* for $(u^{n+\frac{1}{2}})^2$ linearizes the nonlinear term $(u^{n+\frac{1}{2}})^2$ (error $\mathcal{O}(\Delta t^2)$ as in the discretization of u'):

$$(u^{n+\frac{1}{2}})^2 \approx u^n u^{n+1}$$

Arithmetic mean on the linear $u^{n+\frac{1}{2}}$ term and a geometric mean for $(u^{n+\frac{1}{2}})^2$ gives a linear equation for u^{n+1}:

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(u^n + u^{n+1}) + u^n u^{n+1}$$

Note: Here we turned a nonlinear algebraic equation into a linear one. No need for iteration! (Consistent $\mathcal{O}(\Delta t^2)$ approx.)

Newton's method

Write the nonlinear algebraic equation as

$$F(u) = 0$$

Newton's method: linearize $F(u)$ by two terms from the Taylor series,

$$\begin{aligned} F(u) &= F(u^-) + F'(u^-)(u - u^-) + \frac{1}{2}F''(u^-)(u - u^-)^2 + \cdots \\ &\approx F(u^-) + F'(u^-)(u - u^-) = \hat{F}(u)\,. \end{aligned}$$

The linear equation $\hat{F}(u) = 0$ has the solution

$$u = u^- - \frac{F(u^-)}{F'(u^-)}\,.$$

Newton's method with an iteration index

$$u^{k+1} = u^k - \frac{F(u^k)}{F'(u^k)}, \quad k = 0, 1, \ldots$$

Newton's method exhibits *quadratic convergence* if u^k is sufficiently close to the solution. Otherwise, the method may diverge.

Using Newton's method on the logistic ODE

$$F(u) = au^2 + bu + c$$

$$F'(u) = 2au + b$$

The iteration method becomes

$$u = u^- + \frac{a(u^-)^2 + bu^- + c}{2au^- + b}, \quad u^- \leftarrow u$$

Start of iteration: $u^- = u^{(1)}$

Using Newton's method on the logistic ODE with typical math notation

Set iteration start as $u^{n,0} = u^{n-1}$ and iterate with explicit indices for time (n) and Newton iteration (k):

$$u^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}$$

Compare notation with

$$u = u^{-} + \frac{\Delta t(u^{-})^2 + (1-\Delta t)u^{-} - u^{(1)}}{2\Delta t u^{-} + 1 - \Delta t}$$

Relaxation may improve the convergence

- Problem: Picard and Newton iteration may change the solution too much
- Remedy: relaxation (less change in the solution)
- Let u^* be the suggested new value from Picard or Newton iteration

Relaxation with *relaxation parameter* ω (weight old and new value):

$$u = \omega u^* + (1-\omega)u^-, \quad \omega \leq 1$$

Simple formula when used in Newton's method:

$$u = u^- - \omega \frac{F(u^-)}{F'(u^-)}$$

Program `logistic.py`

```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    if choice == 'Picard1':
        choice = 'Picard';  max_iter = 1

    u = np.zeros(Nt+1)
    iterations = []
    u[0] = u0
    for n in range(1, Nt+1):
        a = dt
        b = 1 - dt
        c = -u[n-1]

        if choice == 'Picard':

            def F(u):
                return a*u**2 + b*u + c

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = omega*(-c/(a*u_ + b)) + (1-omega)*u_
                k += 1
            u[n] = u_
            iterations.append(k)
```

Implementation; part 2

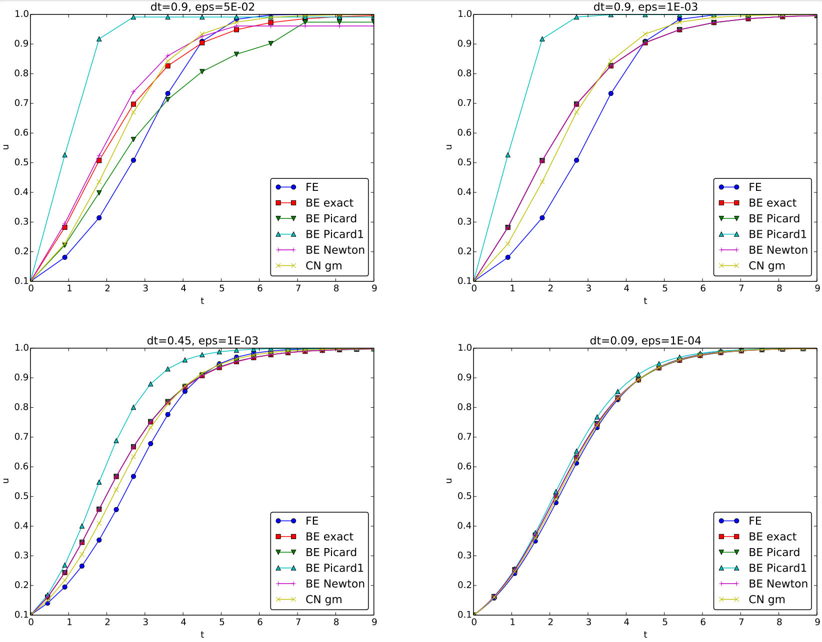
```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    ...
        elif choice == 'Newton':

            def F(u):
                return a*u**2 + b*u + c

            def dF(u):
                return 2*a*u + b

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = u_ - F(u_)/dF(u_)
                k += 1
            u[n] = u_
            iterations.append(k)
    return u, iterations
```

Implementation; part 3

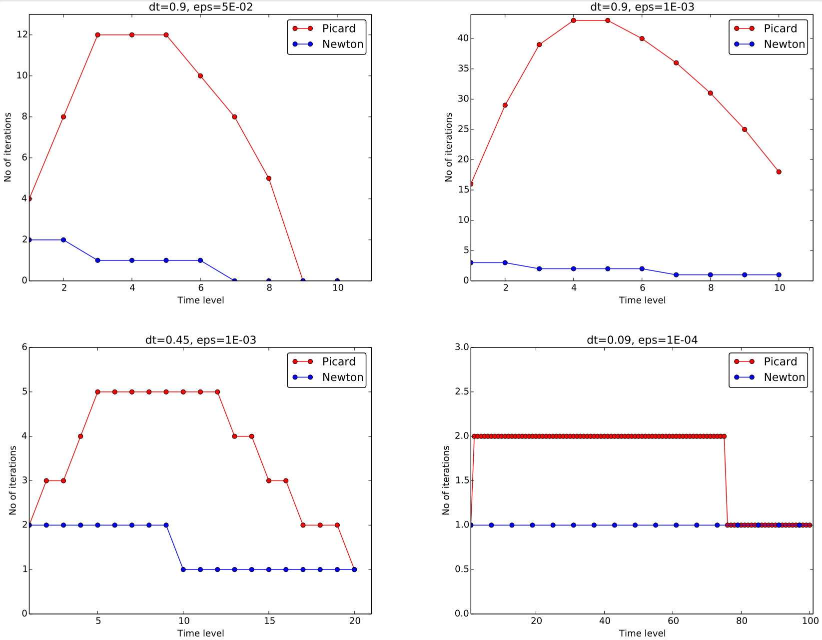
The Crank-Nicolson method with a geometric mean:

```
def CN_logistic(u0, dt, Nt):
    u = np.zeros(Nt+1)
    u[0] = u0
    for n in range(0, Nt):
        u[n+1] = (1 + 0.5*dt)/(1 + dt*u[n] - 0.5*dt)*u[n]
    return u
```

Experiments: accuracy of iteration methods



Experiments: number of iterations



The effect of relaxation can potentially be great!

- $\Delta t = 0.9$: Picard required 32 iterations on average
- $\omega = 0.8$: 7 iterations
- $\omega = 0.5$: 2 iterations (!) - optimal choice

Other $\omega = 1$ experiments:

Δt	ϵ_r	Picard	Newton
0.2	10^{-7}	5	2
0.2	10^{-3}	2	1
0.4	10^{-7}	12	3
0.4	10^{-3}	4	2
0.8	10^{-7}	58	3
0.8	10^{-3}	4	2

Generalization to a general nonlinear ODE

$$u' = f(u, t)$$

Note: f is in general nonlinear in u so the ODE is nonlinear

Explicit time discretization

Forward Euler and all explicit methods sample f with known values and all nonlinearities are gone:

$$\frac{{\color{red}u^{n+1}} - u^n}{\Delta t} = f(u^n, t_n)$$

Backward Euler discretization

Backward Euler $[D_t^- u = f]^n$ leads to nonlinear algebraic equations:

$$F(u^n) = u^n - \Delta t\, f(u^n, t_n) - u^{n-1} = 0,$$

Alternative notation:

$$F(u) = u - \Delta t\, f(u, t_n) - u^{(1)} = 0\,.$$

Picard iteration for Backward Euler scheme

A simple Picard iteration, not knowing anything about the nonlinear structure of f, must approximate $f(u, t_n)$ by $f(u^-, t_n)$:

$$\hat{F}(u) = u - \Delta t\, f(u^-, t_n) - u^{(1)}\,.$$

The iteration starts with $u^- = u^{(1)}$ and proceeds with repeating

$$u^* = \Delta t\, f(u^-, t_n) + u^{(1)}, \quad u = \omega u^* + (1-\omega)u^-, \quad u^- \ \leftarrow \ u,$$

until a stopping criterion is fulfilled.

Manual linearization for a given $f(u, t)$

- $f(u^-, t)$: *explicit* treatment of f (as in time-discretization)
- $f(u, t)$: *fully implicit* treatment of f
- If f has some structure, say $f(u, t) = u^3$, we may think of a *partially implicit* treatment: $(u^-)^2 u$
- More implicit treatment of f often gives faster convergence (as it gives more stable time discretizations)

Computational experiments with partially implicit treatment of f

- $f(u,t) = -u^3$:
  - $(u^-)^3$ linearization: 22, 9, 6 iterations
  - $(u^-)^2 u$ linearization: 8, 5, 4 iterations
- $f(u,t) = e^{-u}$: a trick $f(u^-, t)u/u^-$ has no effect
- $f(u,t) = \sin(2(u+1))$: a trick $f(u^-, t)u/u^-$ has effect (7, 9, 11 iterations vs 17, 21, 20)

Newton's method for Backward Euler scheme

Newton's method requires the computation of the derivative

$$F'(u) = 1 - \Delta t \frac{\partial f}{\partial u}(u, t_n)$$

Algorithm for Newton's method for $u' = f(u, t)$

Start with $u^- = u^{(1)}$, then iterate

$$u = u^- - \omega \frac{F(u^-)}{F'(u^-)} = u^- - \omega \frac{u^{(1)} + \Delta t\, f(u^-, t_n)}{1 - \Delta t \frac{\partial}{\partial u} f(u^-, t_n)}$$

Crank-Nicolson discretization

The standard Crank-Nicolson scheme with arithmetic mean approximation of f reads

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(f(u^{n+1}, t_{n+1}) + f(u^n, t_n))$$

Nonlinear algebraic equation:

$$F(u) = u - u^{(1)} - \Delta t \frac{1}{2} f(u, t_{n+1}) - \Delta t \frac{1}{2} f(u^{(1)}, t_n) = 0$$

Picard and Newton iteration in the Crank-Nicolson case

Picard iteration (for a general f):

$$\hat{F}(u) = u - u^{(1)} - \Delta t\frac{1}{2}f(u^{-}, t_{n+1}) - \Delta t\frac{1}{2}f(u^{(1)}, t_n)$$

Newton's method:

$$F'(u) = 1 - \frac{1}{2}\Delta t\frac{\partial f}{\partial u}(u, t_{n+1})$$

Systems of nonlinear algebraic equations

$$\boxed{\begin{aligned} x\cos y + y^3 &= 0 \\ y^2 e^x + xy &= 2 \end{aligned}}$$

Systems of nonlinear algebraic equations arise from solving *systems of ODEs* or solving *PDEs*

Notation for general systems of algebraic equations

$$F(u) = 0,$$

where

$$u = (u_0, \ldots, u_N), \quad F = (F_0, \ldots, F_N)$$

Special linear system-type structure
(arises frequently in PDE problems):

$$A(u)u = b(u)$$

Picard iteration

Picard iteration for $F(u) = 0$ is meaningless unless there is some structure so we can linearize. For $A(u)u = b(u)$ we can linearize

$$A(u^-)u = b(u^-)$$

Note: we solve a system of nonlinear algebraic equations as a sequence of linear systems.

Algorithm for relaxed Picard iteration

Given $A(u)u = b(u)$ and an initial guess u^-, iterate until convergence:

1. solve $A(u^-)u^* = b(u^-)$ with respect to u^*
2. $u = \omega u^* + (1-\omega)u^-$
3. $u^- \leftarrow u$

"Until convergence": $||u - u^-|| \leq \epsilon_u$ or $||A(u)u - b|| \leq \epsilon_r$

Newton's method for $F(u) = 0$

Linearization of $F(u) = 0$ equation:

$$F(u^-) + J(u^-) \cdot (u - u^-)$$

where J is the *Jacobian* of F, defined by

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

e Approximate the original nonlinear system $F(u) = 0$ by

$$\hat{F}(u) = F(u^-) + J(u^-) \cdot (u - u^-) = 0,$$

which is linear in u

Algorithm for Newton's method

$$\hat{F}(u) = F(u^-) + J(u^-) \cdot (u - u^-) = 0,$$

Solution by a two-step procedure:

1. solve linear system $J\delta u = -F(u^-)$ wrt δu
2. update $u = u^- + \delta u$

Relaxed update:

$$u = \omega(u^- + \delta u) + (1 - \omega)u^- = u^- + \omega\delta u$$

Newton's method for $A(u)u = b(u)$

For

$$F_i = \sum_k A_{i,k}(u)u_k - b_i(u)$$

one gets

$$J_{i,j} = \frac{\partial F_i}{\partial u_j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j} u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}$$

Matrix form:

$$(A + A'u + b')\delta u = -Au + b$$

$$(A(u^-) + A'(u^-)u^- + b'(u^-))\delta u = -A(u^-)u^- + b(u^-)$$

Compare with Picard iteration:

Combined Picard-Newton algorithm

Observation: Newton's method contains all the terms in Picard iteration

Notice

Given $A(u)$, $b(u)$, and an initial guess u^-, iterate until convergence:

1. solve $(A + \gamma(A'(u^-)u^- + b'(u^-)))\delta u = -A(u^-)u^- + b(u^-)$ with respect to δu
2. $u = u^- + \omega\delta u$
3. $u^- \;\leftarrow\; u$

Note:

- $\gamma = 1$: Newton's method
- $\gamma = 0$: Picard iteration

Stopping criteria

Let $||\cdot||$ be the standard Eucledian vector norm. Several termination criteria are much in use:

- Absolute change in solution: $||u - u^-|| \leq \epsilon_u$
- Relative change in solution: $||u - u^-|| \leq \epsilon_u ||u_0||$, where u_0 denotes the start value of u^- in the iteration
- Absolute residual: $||F(u)|| \leq \epsilon_r$
- Relative residual: $||F(u)|| \leq \epsilon_r ||F(u_0)||$
- Max no of iterations: stop when $k > k_{\max}$

Combination of absolute and relative stopping criteria

Problem with relative criterion: a small $||F(u_0)||$ (because $u_0 \approx u$, perhaps because of small Δt) must be significantly reduced. Better with absolute criterion.

- Can make combined absolute-relative criterion
- ϵ_{rr}: tolerance for relative part
- ϵ_{ra}: tolerance for absolute part

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra}$$

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)||+\epsilon_{ra} \quad \text{or} \quad ||\delta u|| \leq \epsilon_{ur}||u_0||+\epsilon_{ua} \quad \text{or} \quad k > k_{\max}\,.$$

Example: A nonlinear ODE model from epidemiology

Spreading of a disease (e.g., a flu) can be modeled by a 2×2 ODE system

$$
\begin{aligned}
S' &= -\beta S I \\
I' &= \beta S I - \nu I
\end{aligned}
$$

Here:

- $S(t)$ is the number of people who can get ill (susceptibles)
- $I(t)$ is the number of people who are ill (infected)
- Must know $\beta > 0$ (danger of getting ill) and $\nu > 0$ ($1/\nu$: expected recovery time)

Implicit time discretization

A Crank-Nicolson scheme:

$$
\begin{aligned}
\frac{S^{n+1}-S^n}{\Delta t} &= -\beta[SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1}) \\
\frac{I^{n+1}-I^n}{\Delta t} &= \beta[SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^nI^n + S^{n+1}I^{n+1}) - \frac{\nu}{2}(I^n + I^{n+1})
\end{aligned}
$$

New notation: S for S^{n+1}, $S^{(1)}$ for S^n, I for I^{n+1}, $I^{(1)}$ for I^n

$$
\begin{aligned}
F_S(S,I) &= S - S^{(1)} + \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + SI) = 0 \\
F_I(S,I) &= I - I^{(1)} - \frac{1}{2}\Delta t\beta(S^{(1)}I^{(1)} + SI) - \frac{1}{2}\Delta t\nu(I^{(1)} + I) = 0
\end{aligned}
$$

A Picard iteration

- We have approximations S^- and I^- to S and I.
- Linearize SI in S ODE as I^-S (linear equation in S!)
- Linearize SI in I ODE as S^-I (linear equation in I!)

$$S = \frac{S^{(1)} - \frac{1}{2}\Delta t\beta S^{(1)}I^{(1)}}{1 + \frac{1}{2}\Delta t\beta I^-}$$

$$I = \frac{I^{(1)} + \frac{1}{2}\Delta t\beta S^{(1)}I^{(1)}}{1 - \frac{1}{2}\Delta t\beta S^- + \nu}$$

Before a new iteration: $S^- \leftarrow S$ and $I^- \leftarrow I$

Newton's method

$$F(u) = 0, \quad F = (F_S, F_I),\ u = (S, I)$$

Jacobian:

$$J = \begin{pmatrix} \frac{\partial}{\partial S} F_S & \frac{\partial}{\partial I} F_S \\ \frac{\partial}{\partial S} F_I & \frac{\partial}{\partial I} F_I \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{2}\Delta t \beta I & \frac{1}{2}\Delta t \beta \\ -\frac{1}{2}\Delta t \beta S & 1 - \frac{1}{2}\Delta t \beta I - \frac{1}{2}\Delta t \nu \end{pmatrix}$$

Newton system: $J(u^-)\delta u = -F(u^-)$

$$\begin{pmatrix} 1 + \frac{1}{2}\Delta t \beta I^- & \frac{1}{2}\Delta t \beta S^- \\ -\frac{1}{2}\Delta t \beta S^- & 1 - \frac{1}{2}\Delta t \beta I^- - \frac{1}{2}\Delta t \nu \end{pmatrix} \begin{pmatrix} \delta S \\ \delta I \end{pmatrix} =$$

$$\begin{pmatrix} S^- - S^{(1)} + \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + S^- I^-) \\ I^- - I^{(1)} - \frac{1}{2}\Delta t \beta (S^{(1)} I^{(1)} + S^- I^-) - \frac{1}{2}\Delta t \nu (I^{(1)} + I^-) \end{pmatrix}$$

Actually no need to bother with nonlinear algebraic equations for this particular model...

Remark:

For this particular system of ODEs, explicit time integration methods work very well. Even a Forward Euler scheme is fine, but the 4-th order Runge-Kutta method is an excellent balance between high accuracy, high efficiency, and simplicity.

Linearization at the differential equation level

Goal: linearize a PDE like

$$\frac{\partial u}{\partial t} = \nabla \cdot (\alpha(u)\nabla u) + f(u)$$

PDE problem

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= \nabla\cdot(\alpha(u)\nabla u) + f(u), && \boldsymbol{x}\in\Omega,\ t\in(0,T]\\
-\alpha(u)\frac{\partial u}{\partial n} &= g, && \boldsymbol{x}\in\partial\Omega_N,\ t\in(0,T]\\
u &= u_0, && \boldsymbol{x}\in\partial\Omega_D,\ t\in(0,T]
\end{aligned}
$$

Explicit time integration

Explicit time integration methods remove the nonlinearity

Forward Euler method:

$$[D_t^+ u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n$$

Written out:

$$\frac{u^{n+1}-u^n}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)$$

This is a *linear equation* in the unknown u^{n+1}, with solution

$$u^{n+1} = u^n + \Delta t\nabla\cdot(\alpha(u^n)\nabla u^n) + \Delta t f(u^n)$$

Disadvantage: $\Delta t \leq (\max\alpha)^{-1}(\Delta x^2 + \Delta y^2 + \Delta z^2)$

Backward Euler scheme

Backward Euler scheme:

$$[D_t^- u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^n$$

Written out:

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n)$$

This is a nonlinear, stationary PDE for the unknown function $u^n(\boldsymbol{x})$

Picard iteration for Backward Euler scheme

We have

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^n)\nabla u^n) + f(u^n)$$

Picard iteration:

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})$$

Start iteration with $u^{n,0} = u^{n-1}$

Picard iteration with alternative notation

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k})$$

- Let's rewrite with a simplified, implementation-friendly notation
- u means the unknown $u^{n,k+1}$
- u^- means the most recent approximation to u
- $u^{(1)}$ means u^{n-1} ($u^{(\ell)}$ means $u^{n-\ell}$)

$$\frac{u - u^{(1)}}{\Delta t} = \nabla \cdot (\alpha(u^-)\nabla u) + f(u^-)$$

Start iteration with $u^- = u^{(1)}$; update with u^- to u.

Backward Euler scheme and Newton's method

- Normally, Newton's method is defined for systems of *algebraic equations*, but the idea of the method can be applied at the PDE level too
- Let $u^{n,k}$ be an approximation to the unknown u^n

We seek a better approximation

$$u^n \approx u^{n,k+1} = u^{n,k} + \delta u$$

- Insert $u^n = u^{n,k} + \delta u$ in the PDE
- Taylor expand the nonlinearities and keep only terms that are linear in δu

Result: linear PDE for the correction δu

Calculation details of Newton's method at the PDE level

Insert $u^{n,k} + \delta u$ for u^n in PDE:

$$\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k} + \delta u)\nabla(u^{n,k} + \delta u)) + f(u^{n,k} + \delta u)$$

Taylor expand $\alpha(u^{n,k} + \delta u)$ and $f(u^{n,k} + \delta u)$:

$$\alpha(u^{n,k} + \delta u) = \alpha(u^{n,k}) + \frac{d\alpha}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx \alpha(u^{n,k}) + \alpha'(u^{n,k})\delta u$$
$$f(u^{n,k} + \delta u) = f(u^{n,k}) + \frac{df}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx f(u^{n,k}) + f'(u^{n,k})\delta u$$

Calculation details of Newton's method at the PDE level

Inserting linear approximations of α and f:

$$
\begin{aligned}
\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = {} & \nabla \cdot (\alpha(u^{n,k}) \nabla u^{n,k}) + f(u^{m,k}) + \\
& \nabla \cdot (\alpha(u^{n,k}) \nabla \delta u) + \nabla \cdot (\alpha'(u^{n,k}) \delta u \nabla u^{n,k}) + \\
& \nabla \cdot (\alpha'(u^{n,k}) \delta u \nabla \delta u) + f'(u^{n,k}) \delta u
\end{aligned}
$$

Note: $\alpha'(u^{n,k})\delta u \nabla \delta u$ is $\mathcal{O}(\delta u^2)$ and therefore omitted.

Result of Newton's method at the PDE level

$$\delta F(\delta u; u^{n,k}) = -F(u^{n,k})$$

with

$$\begin{aligned}
F(u^{n,k}) &= \frac{u^{n,k} - u^{n-1}}{\Delta t} - \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}) \\
\delta F(\delta u; u^{n,k}) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^{n,k})\nabla \delta u) + \\
&\quad \nabla\cdot(\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u
\end{aligned}$$

Note:

- δF is linear in δu
- F contains only known terms

Similarity with Picard iteration

Rewrite the PDE for δu using $u^{n,k} + \delta u = u^{n,k+1}$:

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) \\ + \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u$$

Note:

- The first line is the same PDE as arise in the Picard iteration
- The remaining terms arise from the differentiations in Newton's method

Using new notation for implementation

- u for u^n
- u^- for $u^{n,k}$
- $u^{(1)}$ for u^{n-1}

$$
\begin{aligned}
F(u^-) &= \frac{u^- - u^{(1)}}{\Delta t} - \nabla\cdot(\alpha(u^-)\nabla u^-) + f(u^-) \\
\delta F(\delta u; u^-) &= -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^-)\nabla\delta u) + \\
&\quad \nabla\cdot(\alpha'(u^-)\delta u\nabla u^-) + f'(u^-)\delta u
\end{aligned}
$$

Combined Picard and Newton formulation

$$
\begin{aligned}
\frac{u - u^{(1)}}{\Delta t} = \nabla \cdot (\alpha(u^-)\nabla u) + f(u^-) + \\
\gamma(\nabla \cdot (\alpha'(u^-)(u - u^-)\nabla u^-) + f'(u^-)(u - u^-))
\end{aligned}
$$

Observe:

- $\gamma = 0$: Picard iteration
- $\gamma = 1$: Newton's method

Crank-Nicolson discretization

Crank-Nicolson discretization applies a centered difference at $t_{n+\frac{1}{2}}$:

$$[D_t u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^{n+\frac{1}{2}}\,.$$

Many choices of formulating an arithmetic means:

$$
\begin{aligned}
[f(u)]^{n+\frac{1}{2}} &\approx f(\frac{1}{2}(u^n + u^{n+1})) = [f(\overline{u}^t)]^{n+\frac{1}{2}},\\
[f(u)]^{n+\frac{1}{2}} &\approx \frac{1}{2}(f(u^n) + f(u^{n+1})) = [\overline{f(u)}^t]^{n+\frac{1}{2}},\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \alpha(\frac{1}{2}(u^n + u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = \alpha(\overline{u}^t)\nabla\overline{u}^t]^{n+\frac{1}{2}},\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \frac{1}{2}(\alpha(u^n) + \alpha(u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = [\overline{\alpha(u)}^t\nabla\overline{u}^t]^{n+\frac{1}{2}},\\
[\alpha(u)\nabla u]^{n+\frac{1}{2}} &\approx \frac{1}{2}(\alpha(u^n)\nabla u^n + \alpha(u^{n+1})\nabla u^{n+1}) = [\overline{\alpha(u)\nabla u}^t]^{n+\frac{1}{2}}\,.
\end{aligned}
$$

Solution of nonlinear equations

- Identify the $F(u) = 0$ for the unknown u^{n+1}
- Apply Picard iteration or Newton's method to the PDE
- Identify the sequence of linearized PDEs and iterate